GPU Accelerated Neural Networks With Dynamic Topology

Travis Chung
Project Leader

Shashank Golla
Code Architecture

Suhib Sam Kiswani
Simulation Details

April 24, 2015

Todo list

Remove the todo list and all todos.

Abstract

Neural Networks lend themselves naturally to being parallelized, although certain caveats exist. Furthermore, there exists little work on the benefits of parallelizing networks having a dynamic topology. We demonstrate a parallel implemention of the the Synfire-Growth simulation (developed by Jun and Jin [1]) that runs on the GPU. Despite the dynamic topology of the neural network, our results show that porting the work of [1] to the GPU provided useful runtime performance benefits.

double check this, fill in some actual data as results come in. add a *tiny* bit more.

1 Introduction

The human brain is one of the most complicated and interesting biological systems in nature. Despite our poor understanding of the brain's complex structure, it is common to describe it as a biological computer, consisting of an dense and tangled network of neurons, which serve as pathways for electrical current. This analogy provides a useful starting point for computational models, and is often employed in the machine learning domain.

Neural Networks (or NN's) can be understood most simply as a complete weighted graph, with vertices representing neurons, and weighted edges representing the strength of a neural connection. This model can be enhanced by allowing the weights to change in response to external simuli, representing a dynamic topology of connected neurons.

This paper describes the simulation of neural networks with dynamic topology on the GPU using the model developed by [1] in 2007. We describe how the implementation of [1] was refactored to run on the GPU and take advantage of parallelization. Our work concludes with a cost-benefit analysis of running this simulation on the GPU, and how neural networks naturally lend themselves to the parallel environment.

elaborate and make sure its 100% obvious our implementation is just a refactoring of [1]

2 Background

Primarily, the study the neuroscience is concerned with the behavior of the brain, the way it is structured, the way it learns, how it develops, how it adapts, and changes with respect to stimuli. While understanding of the central nervous system continues to grow, there is so much that is unknown about the brain: the storage and accessing of memories, how the brain retains information, the idea of consciousness, and how sensory input is translated into smell, taste, or pain. The modeling and simulating of brain activity is vital in observing the behavior of the brain and building intuition.

There are generally two families of algorithms for the simulation of neural networks, described in detail by [2]. The two families are synchronous ("clock-driven") or asynchronous ("event-driven") algorithms. In synchronous algorithms neurons are updated only when they receive or emit a spike, whereas "clock-driven" algorithms update all neurons simultaneously at every tick of a clock. There are plenty of simulations using synchronous algorithms, because the spike times are a defined time grid. To get exact simulations of neuron spiking, asynchronous algorithms are recommended. Asynchronous algorithms have been developed for simpler models, but for very large networks, the simulation time and number of spikes becomes problematic for computation.

proposal material. clean up, rephrase statements, and trim the fat.

3 Previous Work

Despite the group's lack of domain knowledge regarding neural networks, the work of [3] provided the initial foundation and direction for our work. The comparisons of the Hodgkin-Huxel (HH) and Izhikevich models of neural networks provided by [3] was a significant aide in deciding which model to use for our simulation. In addition to their analysis, [3] provides evidence for parallel implementations on the GPU that achieve speedups of greater than 110 times their corresponding CPU implementations. Though we could not reproduce a speedup of their magnitude, it gave our work its initial direction.

The model we chose to implement on the GPU was one developed and detailed by [1], suggested by an informal advisor for this project.

In addition to developing the simulation model, it was also implemented by [1]. By using their implementation as our foundation, we were able to run a

requires a clean-up and a couple of revisions to fit the tone of the paper. Include a paragraph on the synfire paper.

be more specific about how [3] served as a foundation for the

faithful representation of their work on the GPU which provides a convenient metric for the efficacy of our work.

clean up. elaborate on implementation maybe?

4 Implementation Details

synfire refactoring description: improvements, complications etc. short and sweet

Algorithm 4.1 Synfire Growth Trial

```
while t < TRIAL_TIME do
    MembranePotential(dt)
    for n ∈ N that spiked do                          ▷ Spike Loop
        Check to see if spiking neuron is saturated.
        Spiking neuron isn't saturated, send spikes along active connections.
        Synaptic Plasticity.
    end for
    for n ∈ N that spiked do                          ▷ Inhibition Phase
        Calculate inhibition.
    end for
    t ← t + dt
end while
SynapticDecay()
```

4.1 On the CPU

depending on time, mention how we tried to help the CPU being competitive in our analysis

4.2 On the GPU

describe some potential + actual optimizations by name; choose 2/3 and go into explicit detail in the subsubsects.

reasons for choosing aforementioned optimizations goes here

4.2.1 Membrane Potential Layer

Due to the fact that each neuron is integrated over dt individually, the mebrane potential layer update step lends itself very naturally to being parallelized. The original implementation of Neur_Dyn function utilized four arrays, all of size 3, to host the results of each progression of the function and eventually update the membrane potential voltages, excitation conductance, and inhibitory conductance. While the array address is stored in the local register, the contents of the arrays are stored in global memory, therefore, each call to the array required roughly 500 clock cycles to execute. Overall, this implementation accessed global memory 92 times.

To reduce global memory access, we utilized three local registers to temporarily hold the membrane potential and conductance's and 3 local registers, in replace of the arrays, each holding an object of type double, to store the results of each progression. After each progression, the temporary voltages and

conductance's are updated. By replacing the arrays with single variables, we had accessed global memory only 3 times. We also eliminated the instance of thread divergence by utilizing the additive identity of the spike voltage. While this required 8 extra floating point operations, the performance benefits of parallel execution, across all threads, far exceeds the extra overhead.

4.2.2 Optimization 2

The Synaptic Decay function required the most time to execute and produced the most overhead per function call. This model we had used required, per trial, the signaling of an inhibitory response through all active and super synaptic connections. This was executed by reducing each synaptic strength measurement with a constant synaptic decay factor. During this process, if the decayed, synaptic strength of each connection failed to exceed the corresponding active or super synaptic threshold, the connection to the post synaptic neuron would be trimmed, and no longer present in either the active or super synaptic network.

The original implementation created two, two-dimensional int arrays, composed of heap memory, to host the post-neuron index of the corresponding active or super synaptic connection. Moreover, the rows corresponded to the index of the pre-neuron, while the column was a pointer to an array of post neurons. When an active or super synaptic connection needed to be trimmed and eliminated from the corresponding network, the program required the allocation of a new array, to house the smaller array of connections, the transfer of elements from the old to the newly allocated array, and the deallocation of the obsolete, heap memory. The constant allocation and deallocation for each active and super synaptic array, per neuron resulted in much overhead. Naturally, since the number of the active and super synaptic structures, for each neuron is different, the implementation is ideally, not suited for execution on the GPU.

To efficiently run the Synaptic Decay model on the GPU, we decided to restructure the Synapse class by creating two static Boolean arrays to house the active and super synaptic connections. Each array was a complete network, of size, network_size-by-network_size, where network_size is the number of neurons existing in the network. If there existed an active or super synaptic connection from one neuron to another neuron, the corresponding index in the corresponding complete network would be flagged as true.

This new implementation is much more suited for execution on the GPU. We greatly reduced the overhead of having to allocate and deallocate the array structure per neuron, by replacing this implementation with one that simply flags the corresponding synaptic connection. Furthermore, since each thread is only required to flag its corresponding thread index, we are able to utilize the Single instruction, multiple thread (SIMT) model, eliminate thread divergence, and eliminate unnecessary work load.

4.2.3 Optimization 3

might be able to get away with only 2 but this serves as a reminder to shoot for the stars

5 Results

IMPORTANT!!

talk about the pros/cons of certain methods and do the major pro/con chart between methods.

be specific about the differences in implementation details. how optimizations can be combined or something to fill up the space

a conclusory sentence that sums up the report

5.1 Optimization 2

Below is the table of the execution times across the restructured CPU code, GPU code, and the original source code. The restructured Synapse class, which utilized two static Boolean arrays to house the active and super synaptic connections, resulted in a speed up of 430x over the original implementation, which required the constant allocation and deallocation of each active and super synaptic array, per neuron. Naturally, this restructured implementation is well suited for execution on the GPU, resulting in a speed up 5.46x over the restructured CPU code, and speed up of 2298.9x over the original implementation.

Synaptic Decay: Execution times	
Platform	**Time** (μs)
GPU	0.4126
CPU	2.2529
Original	969.2

The restructured CPU code is naturally more suited for execution on the GPU. We greatly reduced the overhead of having to allocate and deallocate the array structure per neuron, by replacing this implementation with one that simply flags the corresponding synaptic connection. Since each thread is only required to flag its corresponding thread index, we are able to utilize the Single instruction, multiple thread (SIMT) model, eliminate thread divergence within the Deactivate function, and eliminate the unnecessary work load.

While the new implementation provides considerable speed up over the original implementation, this new approach is limited on the size of the network. Since the size of our network was 1000, the GPU only required 32 MB of space, 16 MB per active and super synaptic network. If the network size was increased to 10,000, then the GPU would require 3.2 GB of space. Therefore, the larger the network, more space of global memory is required on the GPU. If space is exceeded, extra overhead will be required to copy data from host to device and back from device to host.

References

[1] Jun, Joseph K and Jin, Dezhe Z, *Development of neural circuitry for precise temporal sequences through spontaneous activity, axon remodeling, and synaptic plasticity.* PLoS ONE (2007) 2(8): e723. doi: 10.1371/journal.pone.0000723

[2] Brette R, Rudolph M, Carnevale T, et al. *Simulation of networks of spiking neurons: A review of tools and strategies. Journal of computational neuroscience.* 2007;23(3):349-398. doi:10.1007/s10827-007-0038-6.

[3] Fidjeland, A.K., Shanahan, M.P. *Accelerated simulation of spiking neural networks using GPUs.* The 2010 International Joint Conference on Neural Networks (IJCNN). (2010)